A Novel ANFIS Algorithm Architecture for FPGA Implementation

John Darvill
Computing and Technology
Anglia Ruskin University
Cambridge, UK
john.darvill@pgr.anglia.ac.uk

Alin Tisan
Computing and Technology
Anglia Ruskin University
Cambridge, UK
alin.tisan@anglia.ac.uk

Marcian Cirstea
Computing and Technology
Anglia Ruskin University
Cambridge, UK
marcian.cirstea@anglia.ac.uk

Abstract— This paper presents a new architecture for the Adaptive Neuro-Fuzzy Inference System (ANFIS) algorithm targeting FPGA implementation. This new architecture offers higher efficiency and scalability in comparison to the existing methods. The proposed architecture is modeled and simulated using VHDL and is targeted at a Xilinx FPGA. Existing implementation architectures are also modeled and comparisons are drawn between them in terms of both performance and logic utilization. The results show that the new architecture offers a reduction in calculation cycles of around 50% in comparison to the architecture from which it's derived. This increase in calculation speed comes with only a modest increase in logic utilization, specifically a 2.5% increase in look-up table (LUT) usage and a 1.5% increase in flip-flop usage. The new architecture also eliminates scalability issues which can arise in the existing architectures when extra input members are required.

I. Introduction

The field of Artificial Intelligence (AI) has become a popular research topic in recent years due to the potential for application in a range of real world problems, such as voice recognition, data mining, image processing and control systems. Two of the most popular AI based algorithms are Artificial Neural Networks (ANN) and fuzzy logic (FL). The ANN is designed to mimic the neural networks of the brain and as such has an inherent capability to learn, making it a powerful tool for function approximation. On the other hand, FL based algorithms are good at decision making, which means they are popular tool for intelligent control solutions. However the FL algorithm lacks a defined training method, meaning it can be difficult to ensure optimal performance, whilst ANN based systems don't benefit from the ability to make decisions. In order to best exploit the capabilities of both types of algorithm, it is possible to utilize a neural network which is trained to perform as a fuzzy logic controller. Using this approach, it is possible to leverage both the learning capabilities of the ANN and the decision making ability of the FL algorithm. One system which has been shown to achieve the best of best types of AI controller is the Adaptive Neuro-Fuzzy Inference System (ANFIS) [1]. The ANFIS algorithm is an adaptive network which has a similar training scheme to the neural network whilst offering equivalent performance to a fuzzy logic inference system. Although the ANFIS algorithm is a computationally complex algorithm to implement, the advancement of fast and affordable processing has seen the network employed in a wide range of applications such as in [2], [3] and [4].

The ANFIS algorithm has been successfully implemented using both DSPs ([5], [6] and [7]) and digital hardware techniques ([8], [9] and [10]). However, the nature of the algorithm means it is ideally suited to a parallel processing implementation. Whilst modern DSPs typically feature multi core devices, offering a degree of parallel processing, true parallel processing can only be achieved using a hardware approach. For this reason, FPGAs are a popular choice for the implementation of such neural based algorithms ([11]) as they allow for improved throughput in comparison to DSPs.

In this paper, a new digital architecture for the implementation of the ANFIS algorithm is proposed. This new architecture improves scalability, allowing for more membership groups to be utilised without adversely affecting the latency. Additionally, the architecture is designed to utilize as few logical resources as possible. The digital architecture presented in this paper is targeted at a Xilinx Zynq device and is primarily implemented using VHDL, although it is portable to other devices and languages. To show the effectiveness of this new approach, it is compared with some existing digital ANFIS architectures.

The rest of this paper is organized as follows: firstly the ANFIS algorithm is introduced, then a review is undertaken of the existing architecture options in digital hardware, after this the novel architecture is discussed, next the implementation is presented before simulations and hardware tests are presented. Finally conclusions are drawn at the end of the paper.

II. Overview of the ANFIS Algorithm

The ANFIS algorithm was originally introduced in 1993 by J-S Jang [1]. This algorithm is comprised of a five layer feed forward neural network, as shown in Fig. 1. When fully trained, this network exhibits behaviour which is analogous to a Sugeno type fuzzy inference system. The Sugeno inference engine is a universal approximator which is capable of approximating even the most complex of non-linear functions.

This non-linear function approximation capability is therefore inherited by the analogous ANFIS algorithm. When coupled with the relative ease of training, thanks to its learning capability, this makes the ANFIS an attractive option for a number of applications. The ANFIS algorithm has proven particularly popular in the field of intelligent control, due to its decision making abilities, such as in [11] and [12]. In both of these applications the novel architecture discussed in this paper could be utilised. In the rest of this section the five layers of the ANFIS network are discussed in more detail.

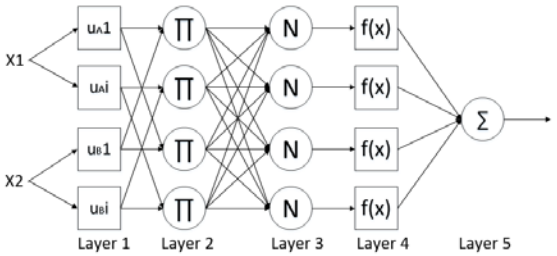

Fig. 1. ANFIS Architecture

A. Layer 1 - Fuzzification

The first stage of the ANFIS converts the crisp input values into fuzzy number sets in much the same way that a fuzzy logic system would. The output of this layer denotes the degree to which the inputs satisfy the membership groups (given as μ_A and μ_B) and has a value of between 0 and 1.

B. Layer 2 and 3 – Firing Strength

The next stage in the system calculates the firing strength of the rules. This is done in two stages - firstly the values of μ_A and μ_B are multiplied together in the second layer. The third layer then performs normalization of the multiplied values, crunching them into a predefined range.

C. Layer 4 – Consequence Parameters

The next stage in the ANFIS calculates the consequence parameters. In a traditional fuzzy system, consisting of IF THEN rules, this part of the algorithm is equivalent to the THEN part of rule. The output of this layer is shown in Eq. (1) with $\overline{w}_i$ denoting he output from the third layer. The output of this stage is effectively the same as the output rule for Takagi-Sugeno type fuzzy inference engine.

$$O_{4,i} = \overline{w}_i f_i$$
$$\text{where } f_i = \text{z} = (px_1 + qx_2 + r) \tag{1}$$

D. Layer 5 – De-fuzzification

The final stage of the ANIFS algorithm is to convert the fuzzy logic sets into a crisp output. This stage takes the simple form of a summation of all the rule outputs, meaning the output is as given in Eq. (2).

$$O_5 = \frac{\sum w_i f_i}{\sum w_i} \tag{2}$$

E. Training the ANFIS

Before the ANFIS can be used in any given application it is firstly necessary to train the underlying neural network. There are two adaptive nodes which must be trained – the input members in layer one and the consequent parameters in the fourth layer. The training method is similar to the normal neural network approach, with the ANFIS being presented with a set of training data. This training set consists of a number of inputs and the expected outputs, with an algorithm being utilised to minimize the error between the expected and actual outputs. The training algorithm is typically a hybrid learning algorithm, featuring forward and backward pass phases. During the forward pass phase of this training the node outputs are fed forward until layer four. At this stage the consequent parameters are tuned using the least squares estimation method. The least squares method is designed to minimize the sum of the squared error of the system output. In the backward pass phase, the membership sets in layer one are tuned. In this phase the error signals are propagated backwards from the output and the membership parameters are optimized using the gradient descent algorithm. This gradient descent algorithm finds the minimum error by moving the membership parameters a distance which is proportional to the functions gradient at a given point. This algorithm is performed in each of the training iterations until the output error is sufficiently minimized.

III. Current State of Art

There are two different types of ANFIS architectures which are presented in research papers [8], [9] and [10] – one using a parallel approach and the other a serial approach. In the parallel architecture, each of the output members has a dedicated logic circuit allowing for all the values to be calculated in a single clock cycle. In comparison, the serial approach has one logic circuit which is used to calculate each of the output members, with the outputs being calculated cyclically. Whilst the parallel architecture offers advantages in the speed of calculation, the logic utilization is considerably greater. The limitations of both of these architectures become exacerbated as more input members are added and the implementation becomes more complex. In the rest of this section both of these architectures are discussed.

A. Serial Architecture

The serial architecture splits the algorithm into four subsystems – a fuzzifier, a permutator, the inference engine and finally a de-fuzzifier as is shown in Fig. 2.

1) Fuzzifier

This block of the architecture relates to layer one in the ANFIS block diagram as shown in Fig. 1 and is responsible for calculating the fuzzy input member values. This block is typically a ROM based look up table (LUT).

2) Permutator

The permutator outputs every possible permutation of input members. This block takes the form of circular shift registers which are shifted in each clock cycle until all permutations have been output. The number of clock cycles required to permutate all data is therefore equal to the total number of permutations. This block represents the main speed bottleneck for this implementation.

3) Inference Engine

The inference engine calculates the values of the numerator and denominator as shown in Eq. (2). This block has three functions which are required in order to calculate these values. The first function is a multiplier which is used to determine the value of the denominator. The second function calculates the value of the consequence parameters, as given by the polynomial equation in Eq. (1). The values of p,q and r are loaded from a ROM, whilst basic adders and multipliers are used to form the rest of the equation. The final function calculates the value of the numerator by multiplying the results of the first two functions.

4) De-Fuzzifier

This final block accumulates the numerator and denominator values output by the inference engine over a full calculation cycle. A divider is then utilised to perform the final operation required to generate the output of the ANFIS as given in (2).

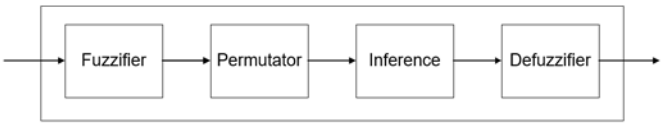

Fig. 2. Serial ANFIS FPGA Implementation

B. Parallel Architecture

The parallel architecture is utilized in [8] and is shown in Fig. 3. This approach is discussed further in this section.

1) Fuzzifier

This block uses a ROM based LUT to perform the fuzzification of the system inputs as in the serial approach.

2) Rule Weight

This block is formed of a single multiplier which performs the calculation shown in Eq. (1). There is one multiplier for each possible permutation of consequence parameters, meaning the amount required is equal to the number of permutations.

3) Consequence Parameters

This subsystem calculates the values for each of the consequence parameters as required for layer 4 in Fig. 1. The method for this is the same as that utilized in the inference engine for the serial approach. However whilst there is one of these circuits in the serial method, in the parallel method there is one used for the calculation of each of the output functions. The total number of circuits required is therefore equal to the total number of possible permutations of the input members. This means that the logic required for this function increases in proportion to the number of input members used which hampers the scalability of this architecture.

4) De-Fuzzifier

The final subsystem in the parallel architecture is responsible for calculating the output and performs three tasks. The first task is to calculate the value of the numerator in Eq. (2) by multiplying each of the rule weights with the related consequence parameter value. The values of all of these multiplications are then added together giving the numerator value. The second task of the de-fuzzifier is to calculate the value of the denominator which is performed through the addition of all the rule weights. The final task of this block is to calculate the algorithm output by dividing these values.

IV. Proposed FPGA Implementation

In the previous section, the two most prevalent ANFIS architectures which are deployed in FPGAs were discussed. Whilst both of these implementations have been designed and implemented for cases with three members, both of these approaches suffer from scalability issues. In the case of the parallel architecture the amount of logic increases in proportion to the number of input members required, meaning excessive logic utilization can become a problem. For the serial approach, the limitation is evident in the number of clock cycles required in the permutator module, which is equal to the number of members squared. In this paper, a novel approach is proposed, which improves the efficiency of the permutator sub system within the serial architecture. This improvement removes redundant permutations, thus reducing the number of clock cycles required by the permutator. This modification improves both the speed and the scalability of the architecture. In the rest of this section, the features of the new architecture are discussed in more detail.

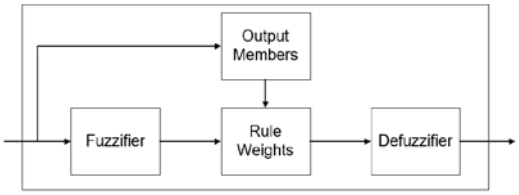

Fig. 3. Parallel ANFIS Implementation

As it was mentioned in the previous discussion of the implementation techniques, the permutator sub system represents something of a speed bottleneck in the serial implementation approach. This bottle neck exists as there is a need to iterate over each permutation of the input members. This means that the total number of iterations required by the permutator is equal to the number of input members squared. This is unnecessary as it is unlikely that any input will ever occupy more than two or three of the possible input members. By way of an example, the application which is simulated in the next section has five input members for each of the two inputs. This means that in the existing serial implementation there would need to be a total of twenty five iterations for the calculation of the permutator. However, in reality one of the inputs can only ever exist in two input members at once, whilst the other can only exist in three members at any one time. This means that the maximum number of iterations required would actually be six. This shows that there are nineteen superfluous operations in this instance when using the current serial method.

Given the inefficiency in the existing implementation, the novel approach seeks to improve the efficiency of the permutator by removing the need to iterate over redundant combinations. In order to facilitate this objective, the shift registers in the original permutator design are replaced with multiplexors. There is then additional logic which is used to drive the address bits of the multiplexor so that each valid combination of outputs is achieved. The full circuit diagram for the new permutator circuit is given in Fig. 4.

The first stage of the novel permutator is to identify which of the input members have a valid, non-zero value. In order for

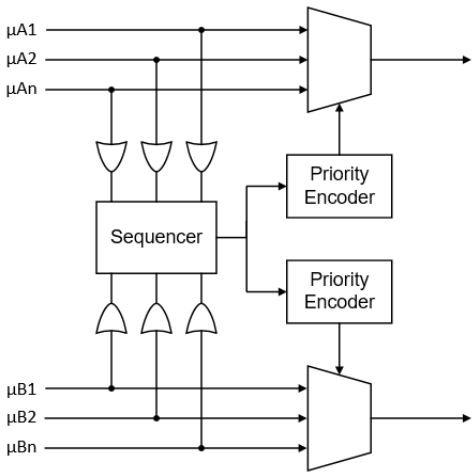

Fig. 4. Novel Permutator Solution

this to be achieved, a member map is created which shows which of the members has a non-zero value. The member map has one bit per input which is set to 1b when the value is non-zero. It is simple to create the logic to drive this map by taking an OR of all the bits in each of the membership values.

Sequencing logic drives the multiplexor address bits, iterating over each of the valid member combinations. The sequencer clears one of the set bits in the member map in each clock cycle. Once all bits are cleared in the first member map, it is restored to the original value and one of the bits is cleared in the second member map. This process is repeated until each of the valid combinations has been iterated over. An example of the sequencer operation for a five input ANFIS is given in TABLE I. The sequencer drives a priority encoder which converts the output into a valid multiplexor address. The output of the priority encoder is a binary representation of the least significant bit which is set in the input. TABLE II. shows the priority encoder process for a 5 bit input.

V. FPGA Implementation

In order to verify the performance of the new architecture for the ANFIS algorithm, a VHDL based model has been developed. This model is derived from the ANFIS-PI based power converter controller solution which was previously presented in [13]. However, this methodology would be equally suited to other ANIFS based industrial electronics applications, such as those presented in [6] and [11]. In order to compare the new method with the existing architectures, models are also developed which utilize those methodologies. All three models have two inputs, both of which utilize five Gaussian input members. This is in comparison to the implementations presented in [8], [9] and [10] which use only three input members. The system has a single 8-bit input, with the delta of this input with respect to time being the second ANFIS input. The models convert the 8 bit inputs to single-precision floating point number and all data is subsequently processed using this precision.

The target device for the implementation will be the Xilinx Zynq, part number XC7Z020, which features an Artix-7 based FPGA core and a dual core ARM processor. The Artix-7 core is a low-end member of the Xilinx family of FPGAs and is chosen to illustrate how the novel architecture can be implemented using simple logic blocks. However, the novel architecture could just as easily be incorporated into more high-performance FPGAs such as the Virtex-7 series. The architecture of the Artix-7 family uses customizable logic blocks (CLB) organized in slices (2 per CLB), each containing 4 lookup tables (LUT) and 8 Flip Flops (FF) that can be used to implement combinatorial and sequential logic circuits.

TABLE I. Sequencer Example

Clock Cycles	x1 Sequence	x2 Sequence
1	01110	01100
2	01100	01100
3	01000	01100
4	01110	01000
5	01100	01000
6	01000	01000

TABLE II. Priority Encoder Logic

In4	In3	In2	In1	In0	Out2	Out1	Out0
0	0	0	0	0	0	0	0
x	x	x	x	1	0	0	1
x	x	x	1	0	0	1	0
x	x	1	0	0	0	1	1
x	1	0	0	0	1	0	0
1	0	0	0	0	1	0	1

The LUTs can also implement ROM or RAM memories, 6 input functions or shift registers. There are also a number of DSP cells which are optimized for fast mathematic operations.

The VHDL models have been synthesized and net lists have been generated for each of the architectures. The logic utilization of the various FPGA cells is shown in TABLE III. These results show that, whilst offering the greatest speed, the parallel implementation requires the greatest amount of logic by far. In addition to this, the long combinatorial chains mean that a large amount of pipelining would needed for the design to operate at higher frequencies. This fact would negate many of the speed gains that are generated from the architecture. This shows that the parallel architecture is only generally suitable for applications which require a greater amount of through put than is achievable with the other methods. The novel approach also requires more logic than the serial implementation although this increase is only modest. There is a 2.5% increase in LUT utilization and 1.5% increase in the FF utilization compared to the serial approach.

VI. Simulation and Implementation Results

The three models are simulated under a number of different scenarios to prove their operation. The output of the models are verified against a simulation model of the algorithm which was created using MATLAB. The ANFIS algorithm developed in MATLAB was created using automated training tools and scripts were then written to export this into a VHDL model. All three of the models have been successfully verified against the MATLAB simulation model. The full sets of simulations are shown in Fig 7 to Fig. 10. The VHDL models have one single bit input to indicate that a new piece of data is available to process, called in_valid in the waveforms. Similarly there is a single bit output which is set high whenever a new output is available, called fis_ready. The 8-bit input to the model is contained in the signal data_in and the output is presented on the signal fis_out.

When carrying out the simulations of the three VHDL models, the main criteria against which the performance is judged is the response time. The metric used for assessing this is the number of clock cycles required per calculation. These results show that the parallel implementation offers the fastest response time, taking a total of 3 clock cycles to generate an output. This speed does come at the expense of a large increase

TABLE III. Logic Utilization

Architecture	Logic Utilization			
	LUT	*FF*	*DSP*	*RAM*
Parallel	56072	373	303	5
Serial	3524	1295	26	6
Novel	3615	1314	26	6

in the amount of logic required for implementation, as previously discussed. This means that this architecture is only suitable for cases which require very high throughput. The novel approach has the next fastest response time with the number of clocks required varying from a minimum of 15 to a maximum of 21, depending on the exact input values. Finally the serial approach is the slowest of the architectures, taking 36 clock cycles to complete a calculation cycle. All of the simulations carried out in this paper have been done using the Xilinx Vivado software suite, version 2016.3.

In order to further verify the novel ANFIS architecture, the VHDL has been committed to silicon and tested. As previously mentioned the hardware target for the model is the FPGA core in a Xilinx Zynq XC7Z020 device. The model has been tested by creating an on-board stimulus generator which emulates the stimulus generated in the VHDL test bench. In order to test the hardware target an on-board scope was added using a Xilinx IP core. The on-board scope is capable of capturing signals on the FPGA and outputting this to a PC using the JTAG interface. The waveforms, which are captured using the Xilinx Vivado software tool, are shown in Fig 11 and Fig 12.The hardware test results show that the hardware implementation matches with the previously verified VHDL model as shown in Fig 9 and Fig 10. Note that the fis_ready signal in the simulations

Another important consideration which is not illustrated in the simulations is the scalability of the different architectures. If the implementation of the ANFIS algorithm was changed to include more than five members then this would have an adverse affect on the existing architectures. In the case of the parallel architecture, this would result in a large increase in the logic utilization. As an example, consider a new implementation using seven input members rather than five. This would mean that there would be a total of 49 consequence parameters rather than 25 in the current implementation. This means that there would be a requirement for nearly double the has been replaced by the anfis_done signal in hardware

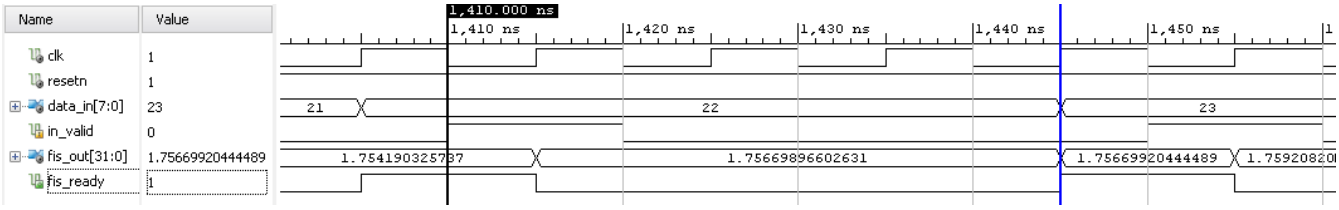

Fig.7. Parallel Implementation Simulation Result

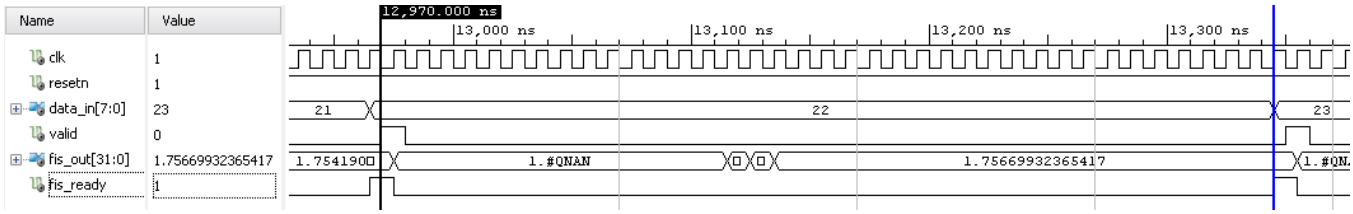

Fig.8. Serial Implementation Simulation Result

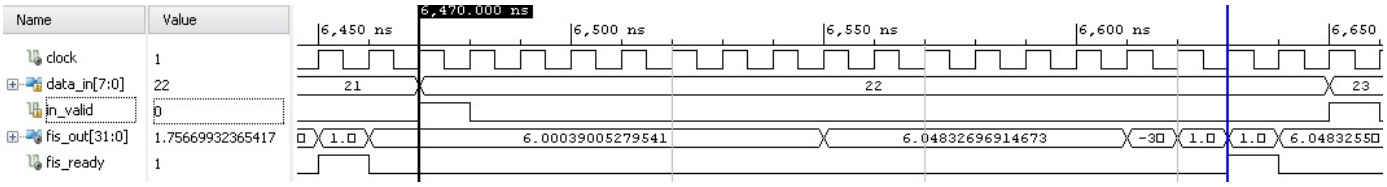

Fig. 9. Novel Implementation Simulation Result with 14 Clock Cycles

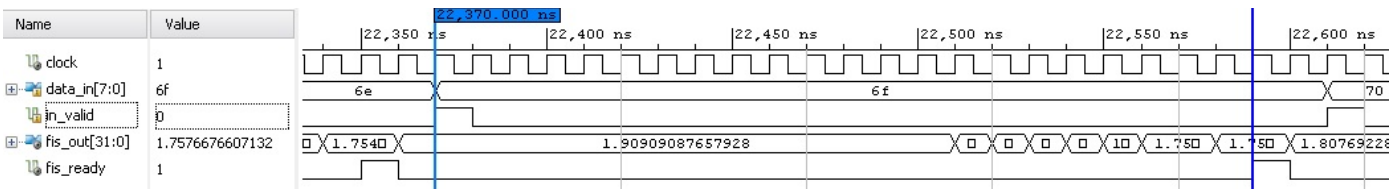

Fig. 10. Novel Implementation Simulation Result with 20 Clock Cycles

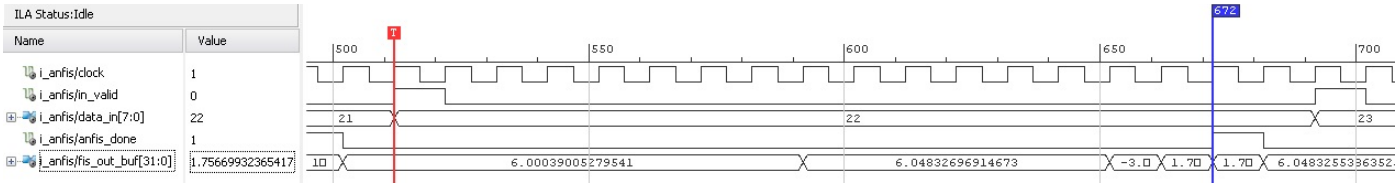

Fig. 11. Novel Architecture Hardware Test Result with 15 Clock Cycles

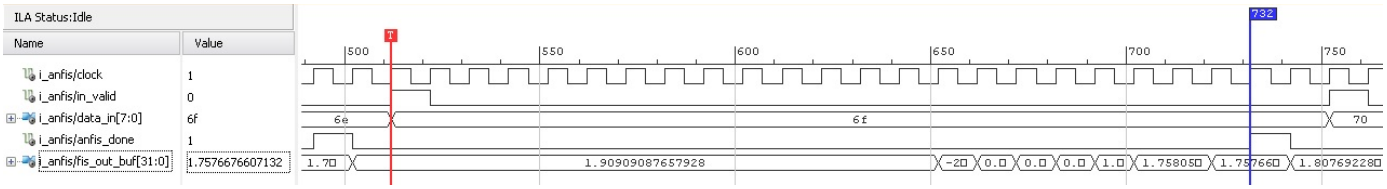

Fig. 12. Novel Architecture Hardware Test Result with 21 Clock Cycles

amount of logic to implement the consequence parameters. Whilst the serial architecture doesn't suffer with excessive increases in logic, calculation speed is affected by the functionality of the permutator circuit. In the five input member implementation a total of 25 permutations are possible. In comparison the seven member implementation would have a total of 49 possible permutations – an increase of 24. As the permutator must output every valid permutation, this would require an increase of 24 clocks for the calculation time. This would take the total number of clocks required per calculation from 37 up to 61. This shows that as the ANFIS implementation is made more complex by adding input members, the speed of the serial implementation decreases.

The novel architecture, on the other hand, suffers from neither of these limiting factors when the number of input members is increased. As the novel implementation is based upon the serial implementation, there would be a similarly modest increase in logic utilization. Due to the efficiency improvements in the permutator, the number of extra clock cycles required would also be significantly less than the serial architecture would require. It is difficult to put an exact figure on this for the novel architecture as it is very much implementation specific. In order to give an illustration, the algorithm presented in this paper can be quickly retrained in MATLAB to use seven input members. In this case the distribution of the input members is very similar to the five input implementation. As a result the novel approach would still only require between 15 and 21 clock cycles for the seven input implementation. This is a good illustration of the improved scalability in comparison to the serial approach.

VII. Conclusion

A novel digital system architecture for the ANFIS algorithm is presented in this paper. This method improves upon the performance of the existing serial architecture by removing redundant calculation cycles. Models of the new architecture, as well as two existing architectures, were then created using VHDL. By simulating the models it was observed that the novel methodology offers improvements in calculation speed compared to the existing serial methodology on which it is based. The novel architecture is shown to require around half the number of clock cycles to perform the calculation. In addition to this, only a modest increase is required in the logic utilization – just a 2.5% increase in the number of LUTs and 1% more FFs. Whilst a parallel architecture was also simulated and shown to offer a faster response time, this came at the expense of using more than ten times as much logic in the target FPGA. This means that this parallel approach is only suitable for the most time sensitive applications. As an additional benefit, the novel approach also allows for greater scalability than the serial approach due to the removal of redundant calculation cycles. A case study was considered with two extra input members and through analysis it can be illustrated that the novel approach would not require any extra calculation time for this. In comparison, the serial approach would require an additional 24 clock cycles.

References

[1] J-S, Jang , "ANFIS: Adaptive-Networtk-Based Fuzzy Inference System." *IEEE Transacrtions on Systems, Man and Cybernetics*, Vol. 23, Iss 3, pp. 665-685, 1993

[2] C.N. Pitas, D.E. Charilas, A.D. Panagopoulos and P. Constantinou, "Adaptive Neuro-Fuzzy Model for Speech and Voice Quality Prediction in Real-World Communication Networks." *IEEE Wireless Communications*, Vol. 20, Iss. 3, pp 1536 – 1284, 2013

[3] C. Chen, B. Zhang, G. Vachtsevanos and M.Orchard, "Machine Condition Predicition Based on Adaptive Neuro-Fuzzy and high-Oder Particle Filtering. "*IEEE Transactions on Industrial Electronics* Vol. 58, Iss. 9, pp. 4353-4364, 2011

[4] J.P.S. Catalao, H.M.I Pousinho and V.M.F Mendes, "Hybrid Wavelet-PSO-ANFIS Approach for Short Term Electricity Prices Forecasting." *IEEE Transactions on Power Systems*, Vol 26, Iss. 1, pp 137-144, 2011

[5] H. Abu-Rub, A. Iqbal, S. Mon Ahmed, F.Z. Peng and G. Baoming, "Quasi-Z-Source Inverter Based Photovoltaic Generation System with Maximum Power Tracking Control using ANFIS." *IEEE Transactions on Sustainable Energy*, Vol 4, Iss. 1, pp 11-20, 2012

[6] M. Singh and A.Chandra, "Real Time Implementation of ANFIS Control for Renewable Interfacing Inverter in 3P4W Distribution Network." *IEEE Transactions on Industrial Electronics*, Vol. 60, Iss. 1, pp. 121-128, 2012

[7] A. Chikh and A. Chandra, "An Optimal Maximum Power Point Tracking Algorithm for PV Systems with Climatic Parameters Estimation." *IEEE Transactions on Sustainable Energy*, Vol. 6, Iss.2, pp. 664-652,2015

[8] F. Gomez-Castaneda, G.M. Tornez-Xavier, L.M. Flores-Nava, O. Arellano-Cardenas and J.A. Moreno-Cadenas, "Photovoltaic Panel Emulator in FPGA Technology Using ANFIS Approach." *International Conference on Electrical Engineering, Computing Science and Automatic Control*, pp 1-6, 2014

[9] H.J.S Saldana and C. Silva-Cardenas, "A Digital Hardware Architecture for a Three-Input One-Output Zero-Order ANFIS." *Third Latin American Symposium on Circuits and Systems*, pp. 1-4, 2012

[10] P.R. Pande, P.L. Paikrao and D.S. Chaudhari, "Digital ANFIS Model Design." *International Journal of Soft Computing (IJSCE)*, vol 3, iss 1, pp 314 - 318, 2013

[11] A.R.Omondi and J.C. Rajapakse, "FPGA Implementations of Neural Networks." Dordrecht, The Netherlands: Springer, 2006

[12] P. Garcia. C.A. Garcia, L.M. Fernandez, F. Lorens and F. Jurado, "ANFIS Based Control of Grid Connected Hybrid System Integrating Renewable Energies, Hydrogen and Batteries." *IEEE Transactions on Industrial Informatics*, Vol. 10, Iss. 2, pp 1107 – 1117, 2013

[13] L. Wang and D.N. Truong, "Stability Enhancement of a Power System with PMSG-Based and a DFIG-Based Offshore Wind Farm Using a SVC With an Adaptive Network Based Fuzzy Inference System." *IEEE Transactions on Industrial Electronics*, Vol. 60, Iss. 7, pp 2799 – 2807, 2013.

[14] J. Darvill, A. Tisan and M. Cirstea, "An ANFIS-PI Based Boost Converter Control Scheme." *IEEE International Conference on Industrial Informations*, pp 632 – 639, 2015.